Report Of Project

Operating system and version:

Windows 10

Graphics hardware:

NVIDIA GeForce MX150

Steps to build:

1. Open opengl.sln

2. Right click on the project name, then go to Properties.

In the Properties Pane, go to "Debugging", and in this pane is a line for "Command-line arguments". Add the name of the ".json" file you would like to test on this line.

3. Run

Rendering time:

c.json: 4 minutes and 2 seconds.

d.json: 2 minutes and 40 seconds.

e.json: 7 minutes and 29 seconds.

f.json: 2 minutes and 42 seconds.

g.json: 8 minutes and 38 seconds.

h.json: 3 minutes and 20 seconds.

i.json: 3 minutes and 15 seconds.

j.json: 5 minutes and 1 seconds.

k.json: 4 minutes and 5 seconds.

l.json: 5 minutes and 13 seconds.

n.json: 2 minutes and 30 seconds.

o.json: 2 minutes and 18 seconds.

Feature #1: Acceleration:

**Feature selected:**

octrees.

**Implementation:**

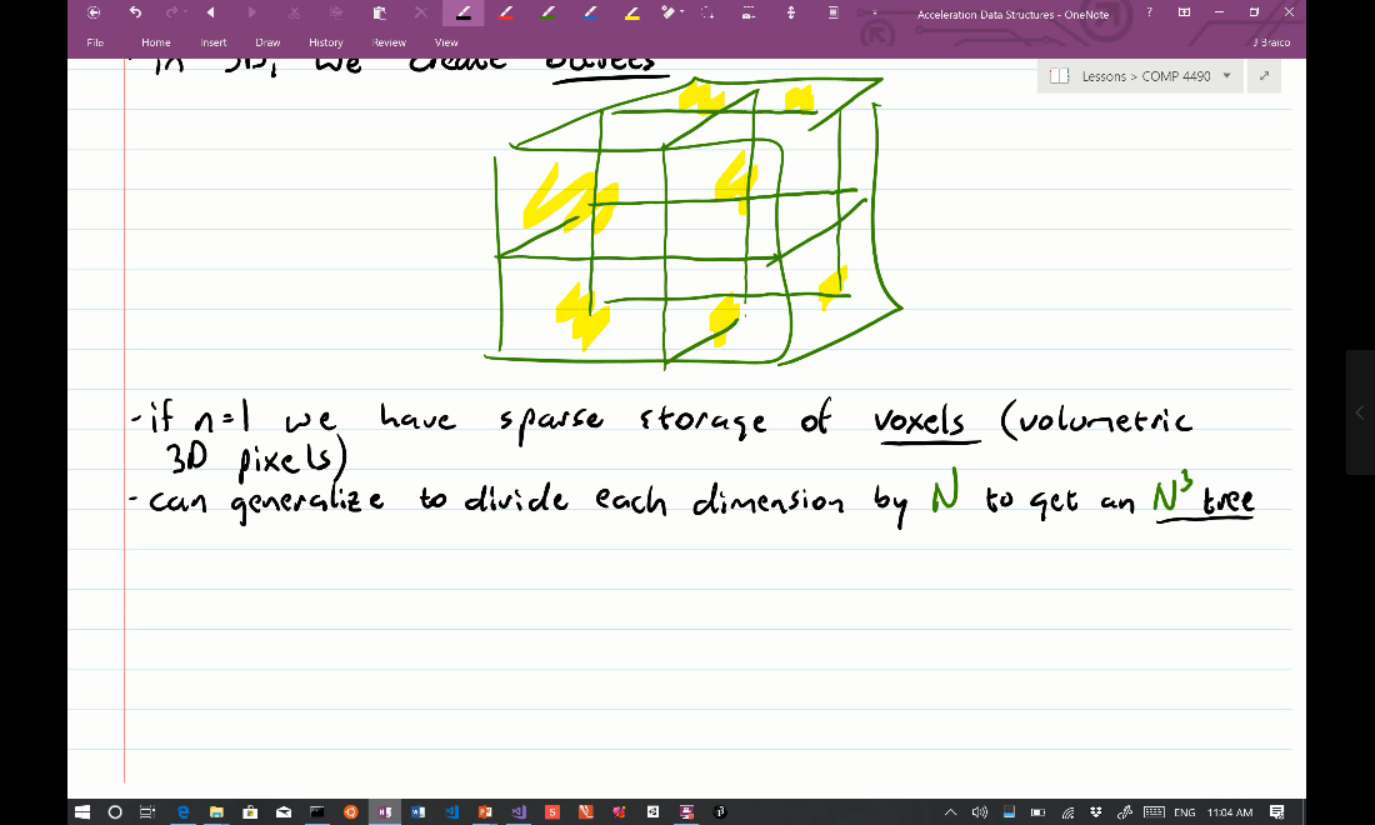
1. Getting the overall bounding of the whole tree.

For each input objects, the program calculates its bounding. Then among all these boundings, the minimums and maximums for x, y and z are found respectively, which form the overall bounding of the whole tree.

I don’t put planes in the octree because they are infinite. It would be even slower if I put them in than not doing that, because each time the program divides a cell, it has to consider the planes.

1. Dividing each cell.

The program divides each cell into 4 sub-cells if there are more than one objects in it and it is bigger than 0.2 in at least one of x, y and z axis. After that, if a sub-cell has any objects in it, the sub-cell is added into the tree, otherwise it is ignored.



(COMP4490-03\_25.mp4)

1. Ray-tracing using the octree.

When doing ray-tracing, the program finds all cells in the octree that are hit by the ray, then all objects contained in those cells are found. The next step is detailed hit-testing between the ray and each of the objects.

**Interaction:**

User could click the display window by mouse. If the click hits a cell in the octree, information about the cell would be printed in the console, including the bounding of the cell and the objects contained in the cell. If the click doesn’t hit any cell, nothing will be printed.

**Outcome:**

After applying octree, the rendering time is more than 80 times faster than before. Now it costs **3 minutes** to render i.json, while that was **270 minutes** before applying octree.

**How to test:**

Please run any input .json file and click on the display window to test. I recommend to use **i.json**, because it generates observable speed promotion.

Feature #2: Transformations

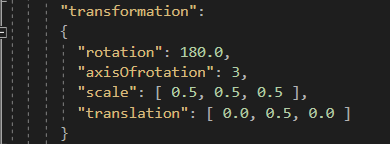
**Feature selected:**

add a transformation to meshes that will be applied before rendering.

**Implementation:**

1. Modifications to c.json.

I modified c.json to generate a new o.json. The modification is that I added a block of “transformation” into it, which contains the coefficients about the transformation the mesh will perform, as shown below.

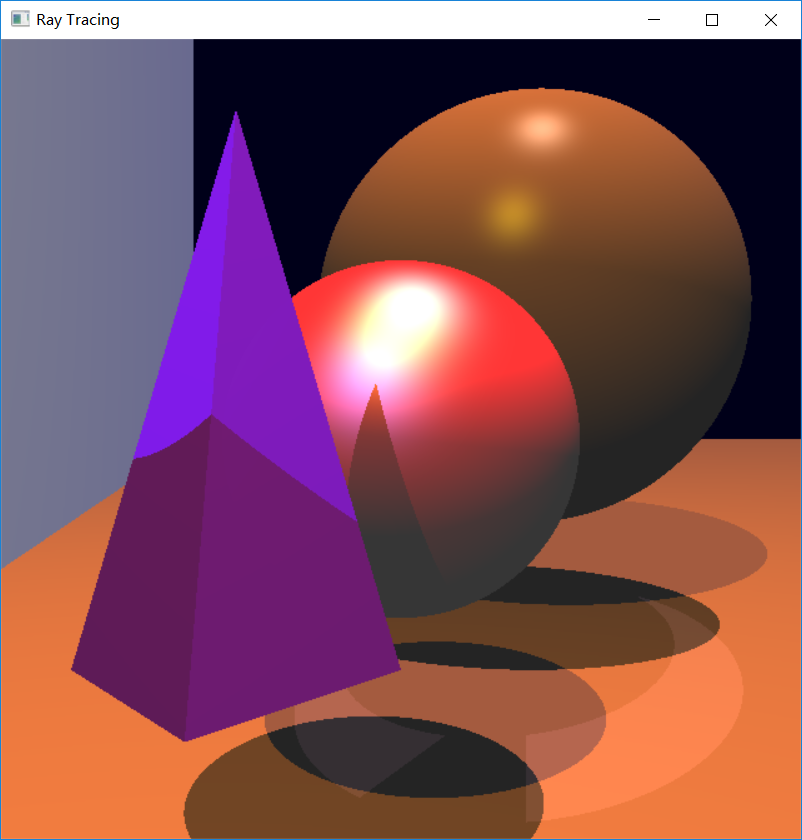


1. Modifications to main program.

The program reads in the coefficients shown above. Then it calculates the barycenter of the mesh by adding all x, y and z values together and calculates their average. Then a translation vector is calculated by using the barycenter to minus the origin, then all vertices are translated based on this vector, after which the mesh’s barycenter and the origin are coincided. Then we do the transformation to all the vertices, after which we translate them back based on the negation of the translation vector mentioned above.

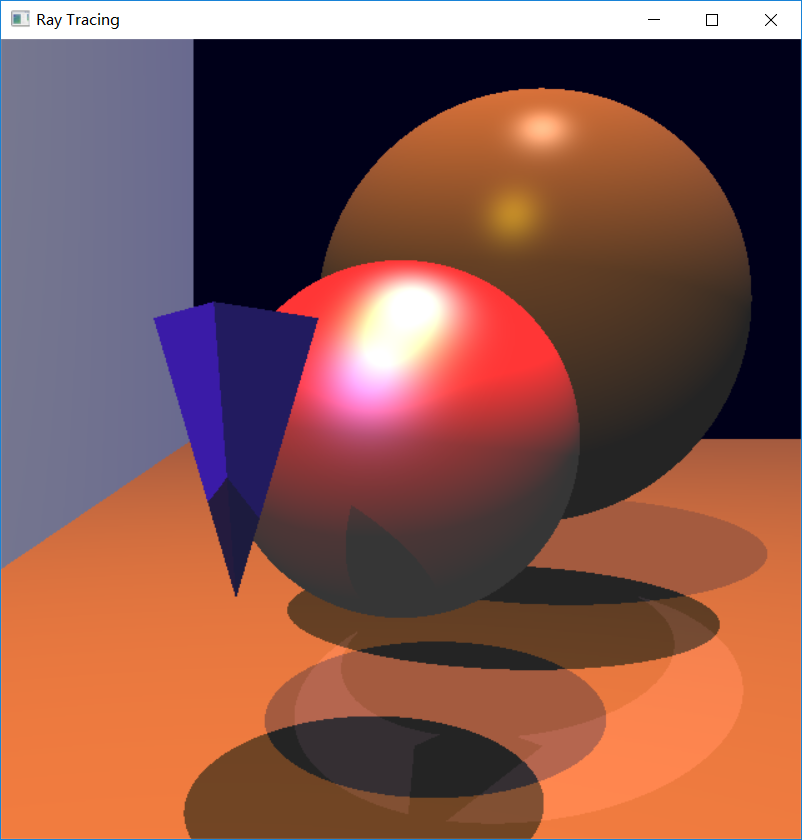
**Outcome:**

In o.json, the mesh is rotated by 180 degree in z axis, is scaled by 50% in all x, y and z axis, and is translated by 0.5 in y axis.



**Before** the transformation

(project\Feature2\_before\_transformation\_c.json.png)



**After** the transformation

(project\Feature2\_after\_transformation\_o.json.png)

**How to test:**

Please run o.json.

Feature #3: Improved Quality

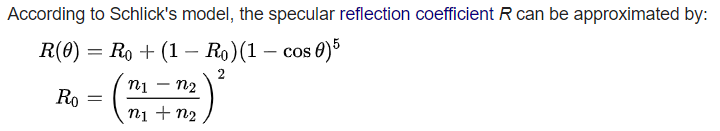
**Feature selected:**

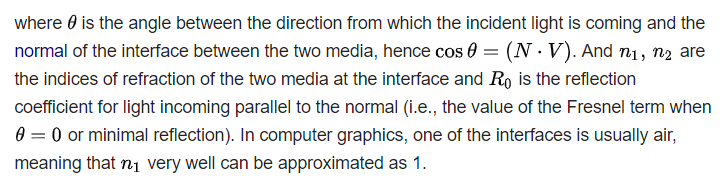
1. Schlick's Approximation for refraction.
2. Oren-Nayar diffuse reflectance model.

**Implementation:**

1. Schlick's Approximation for refraction.

The program calculates a new reflective coefficient using the formula shown below, and uses it to replace the original reflective coefficient of each object to simulate the mirror reflection.



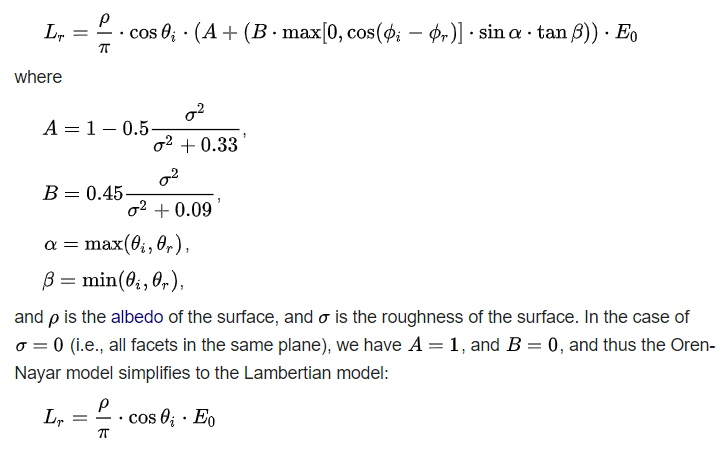


(<https://en.wikipedia.org/wiki/Schlick%27s_approximation>)

Besides, the program uses (1-R(θ)) as the transmission coefficient of each object to simulate the transmission and refraction.

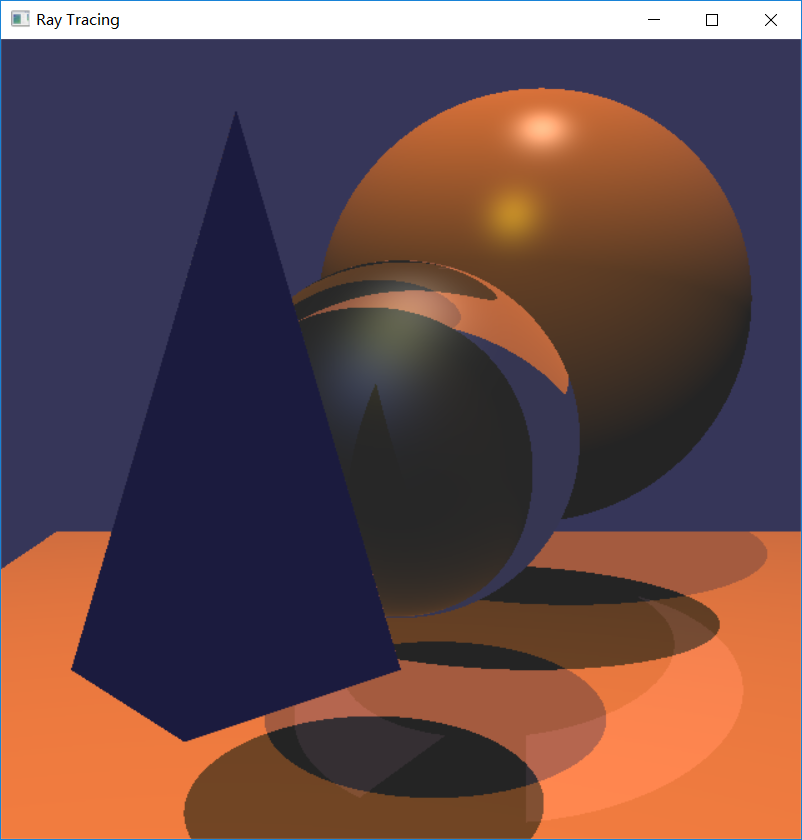
1. Oren-Nayar diffuse reflectance model.

The program calculates the intensity of diffuse using the formula shown below,



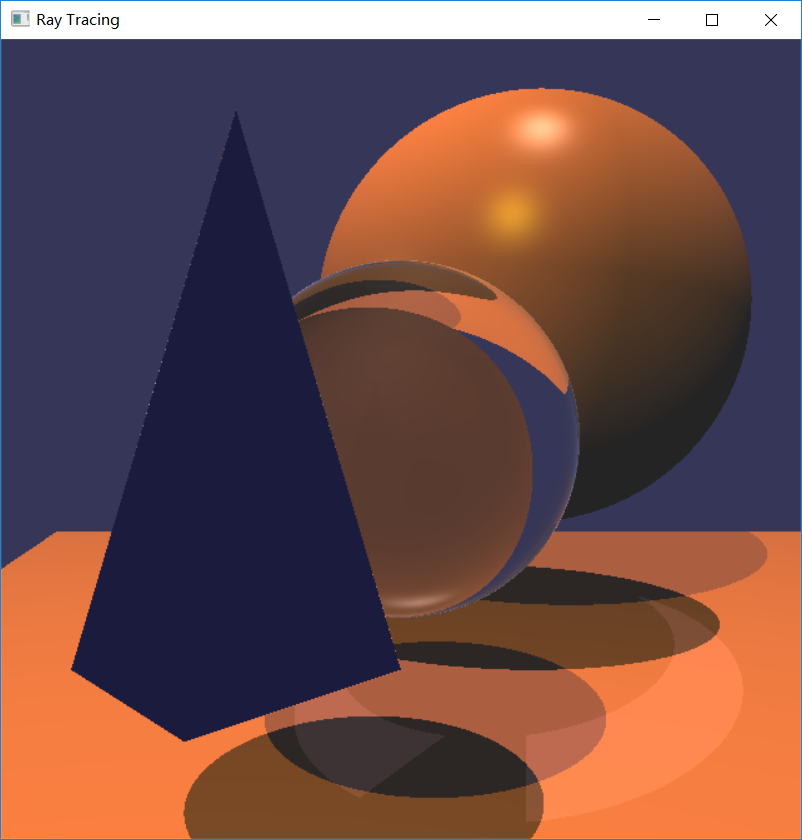
(https://en.wikipedia.org/wiki/Oren%E2%80%93Nayar\_reflectance\_model)

**Outcome:**



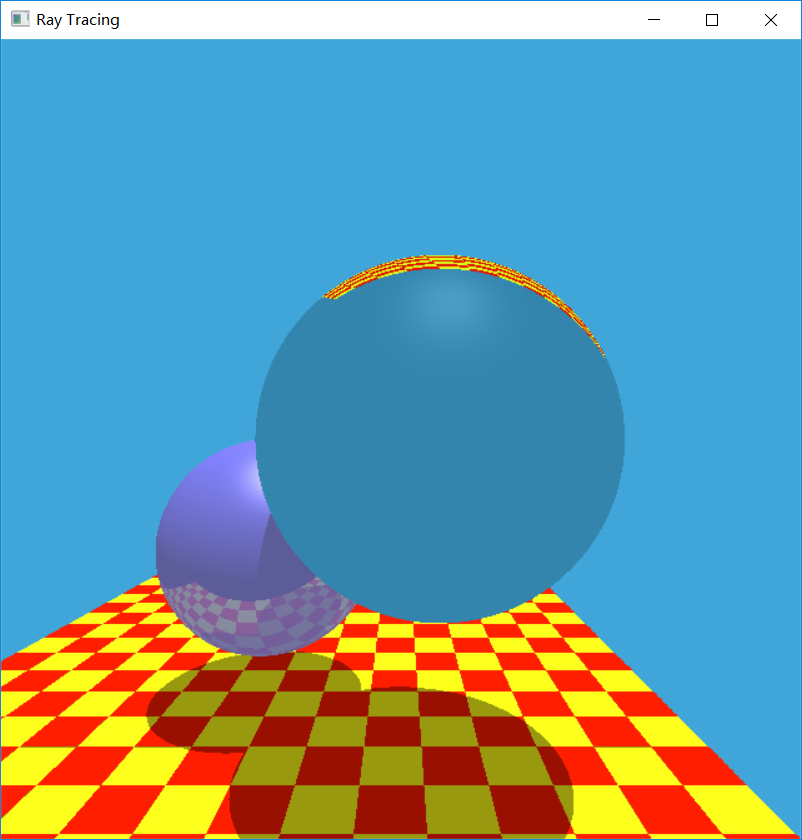
**Before** applying Schlick's Approximation for refraction.

(g\_json\_origin.png)



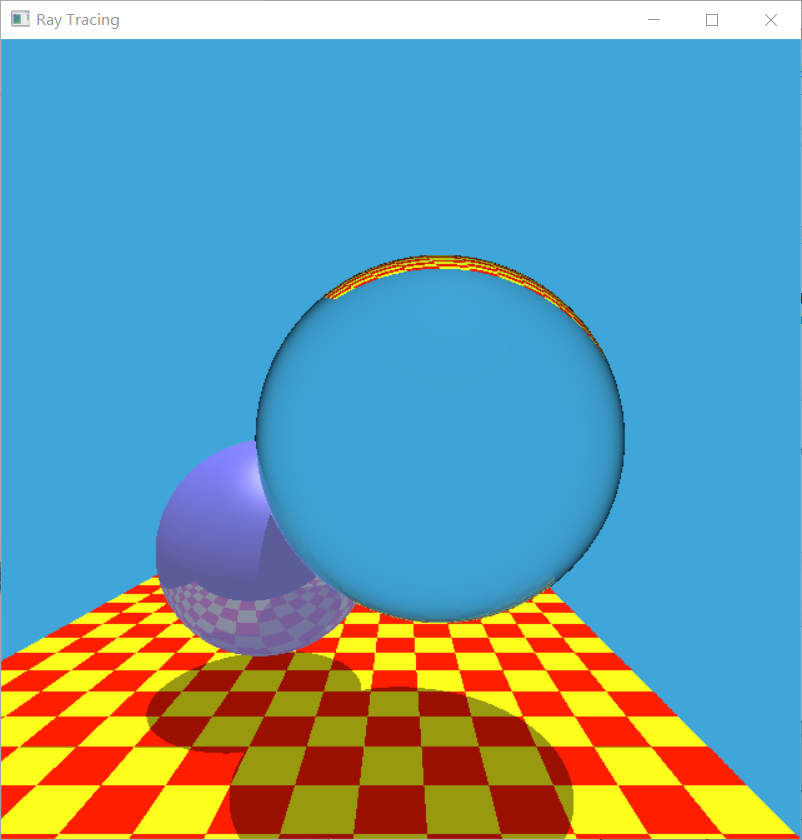
**After** applying Schlick's Approximation for refraction, the sphere in the middle looks better.

(Feature3\_Schlick's Approximation\_g.json.png)



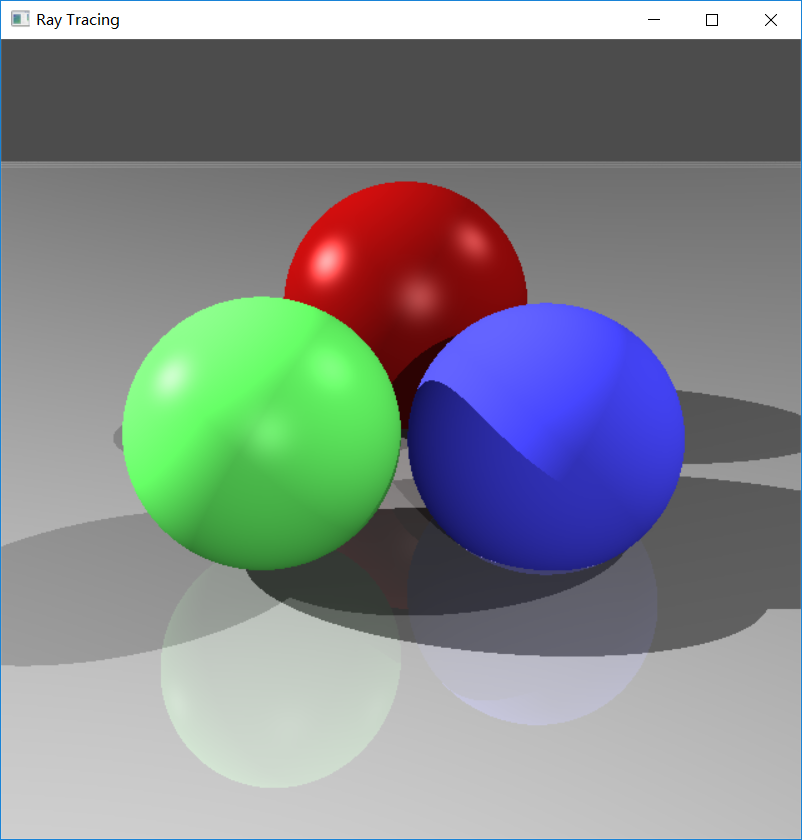
**Before** applying Schlick's Approximation for refraction

(i\_json\_origin.png)



**After** applying Schlick's Approximation for refraction, the sphere in the front looks better.

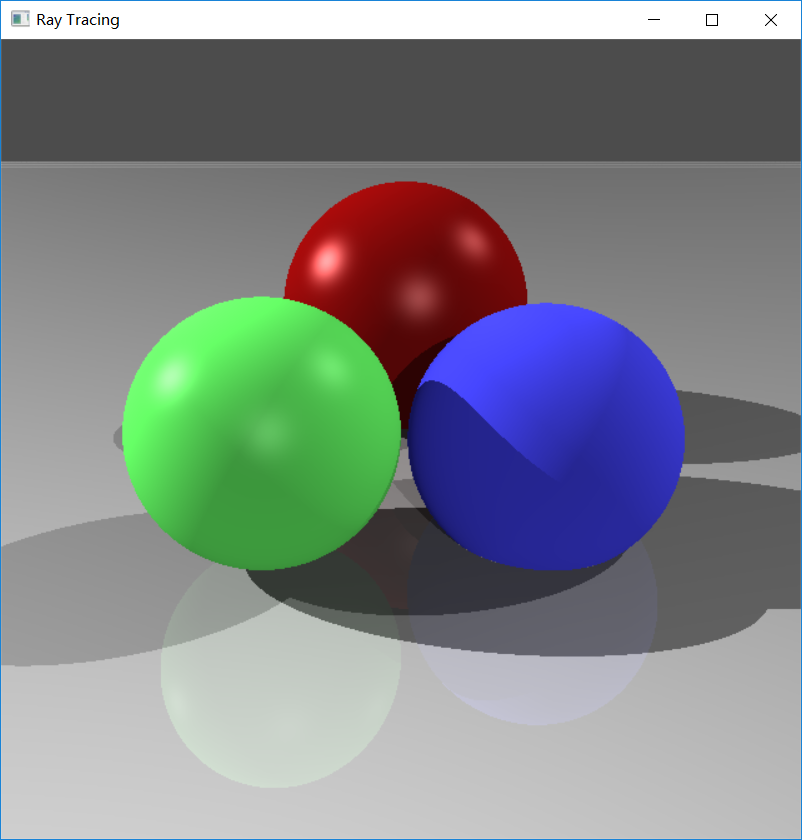
(Feature3\_Schlick's Approximation\_i.json.png)

****

**Before** applying Oren-Nayar diffuse reflectance model.

(Roughness coefficient: **0.0**)

(d\_json\_origin.png)

****

**After** applying Oren-Nayar diffuse reflectance model, the 3 spheres reflect less light.

(Roughness coefficient: **1.0**)

(Feature3\_Oren-Nayar\_n.json.png)

**How to test:**

1. Schlick's Approximation for refraction

Please run g.json and i.json to see the output after applying.

1. Oren-Nayar diffuse reflectance model

Please run d.json to see the output before applying and run n.json to see that after applying.

Feature #4: More Geometry

**Feature selected:**

CSG objects using union, intersection, and difference of other objects.

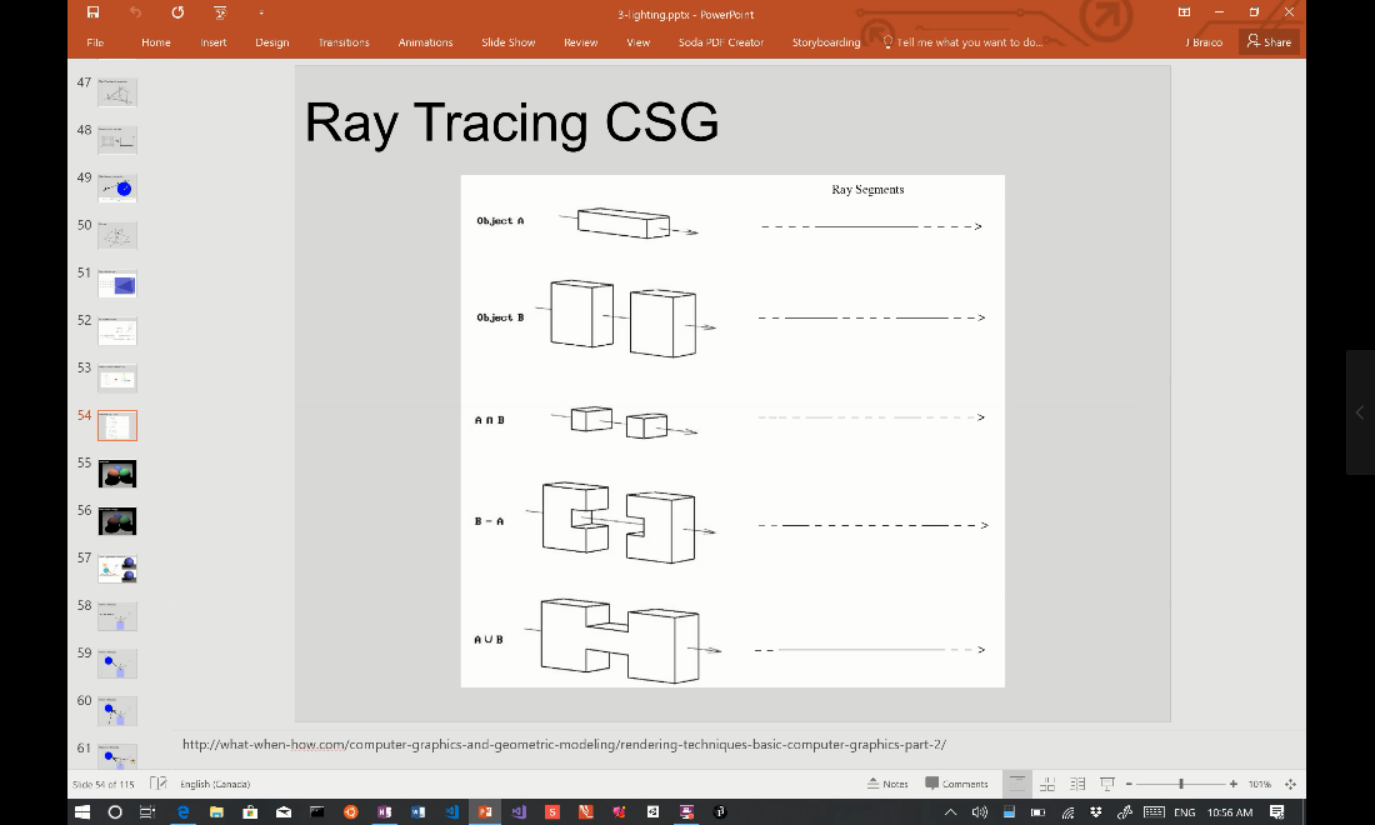
**Implementation:**

The program implements union, intersection, and difference of 2 spheres. To achieve that, new strategies of hit-testing are added. For

1. Union: We consider the ray hits the object if it hits **either** the 2 spheres.

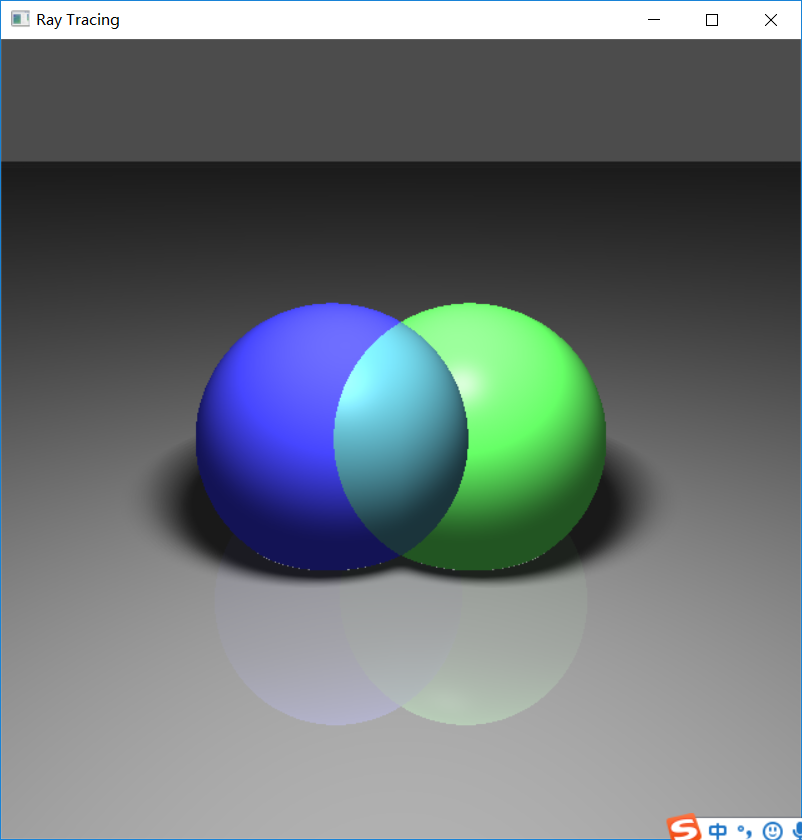
2. Intersection: We consider the ray hits the object if it hits **both** the 2 spheres.

3. Difference: We consider the ray hits the object if it hits the first sphere but not the second sphere.



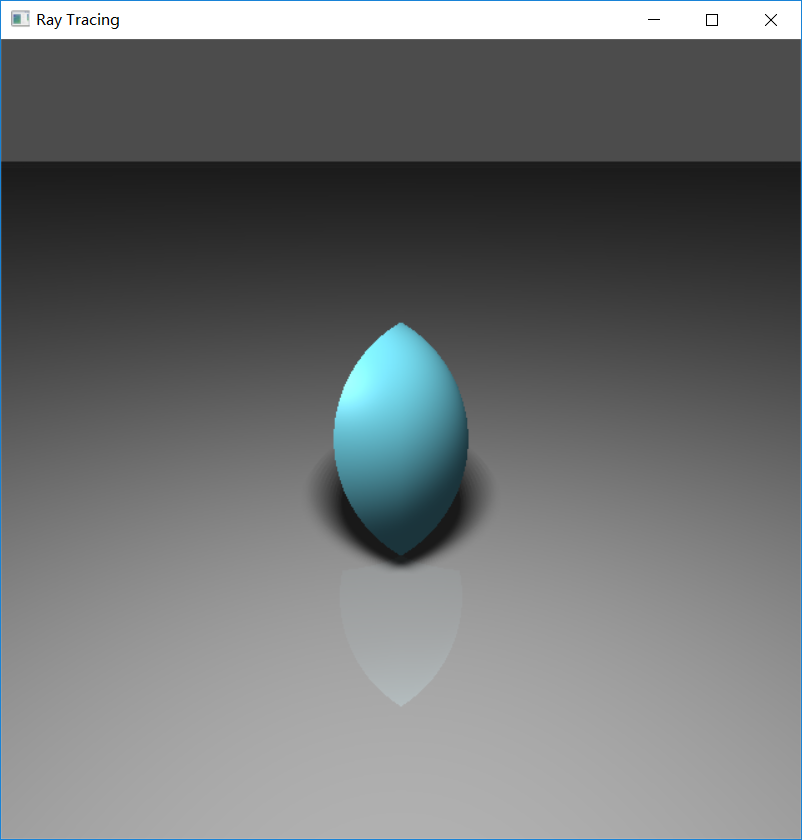
(COMP4490-03\_13.mp4)

For union and intersection, the coefficients, such as reflective and refraction, of the overlapping part come from the average of the 2 spheres.

**Outcome:**

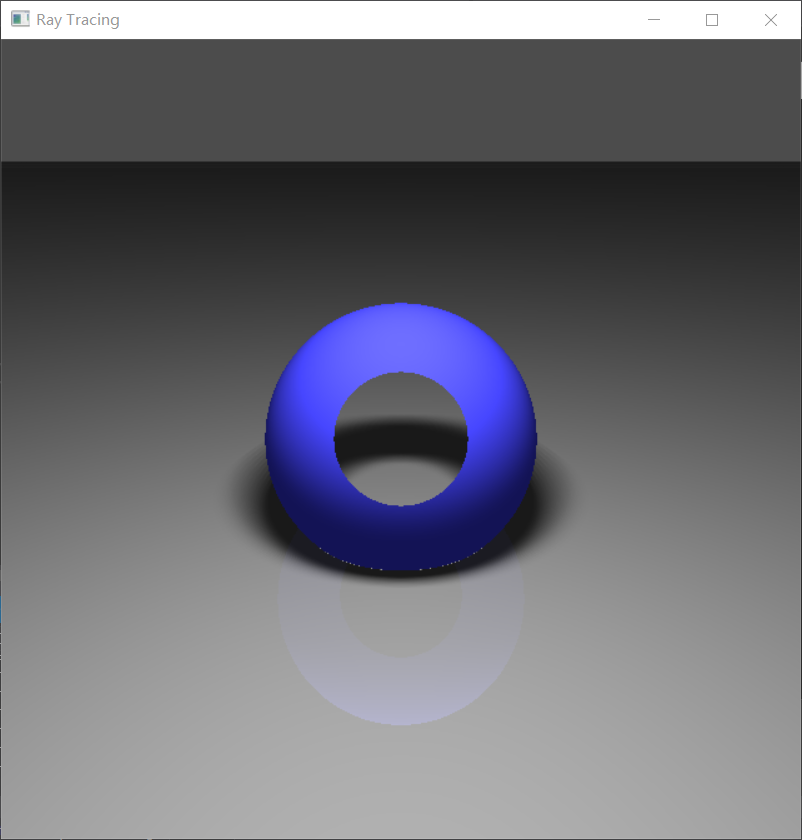
Union of 2 spheres

(Feature4\_union\_j.json.png)

****

Intersection of 2 spheres

(Feature4\_intersection\_k.json.png)

****

Difference of 2 spheres

(Feature4\_difference\_l.json.png)

**How to test:**

1. Union:

please run j.son.

1. Intersection:

please run k.json.

1. Difference:

please run l.json.

Feature #5: Techniques

**Feature selected:**

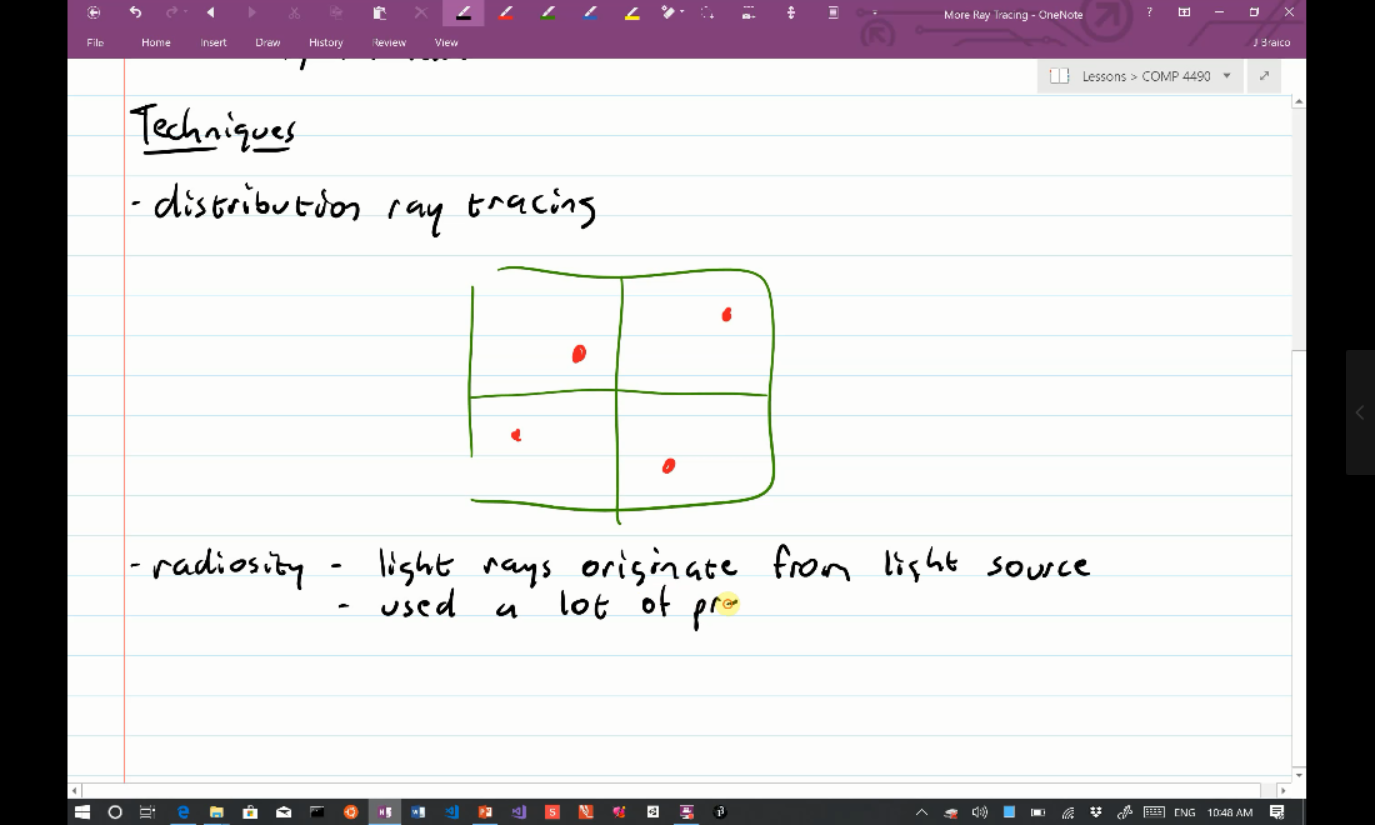
Distribution ray tracing:

1. Antialiasing.
2. Area lighting.

**Implementation:**

1. Antialiasing.

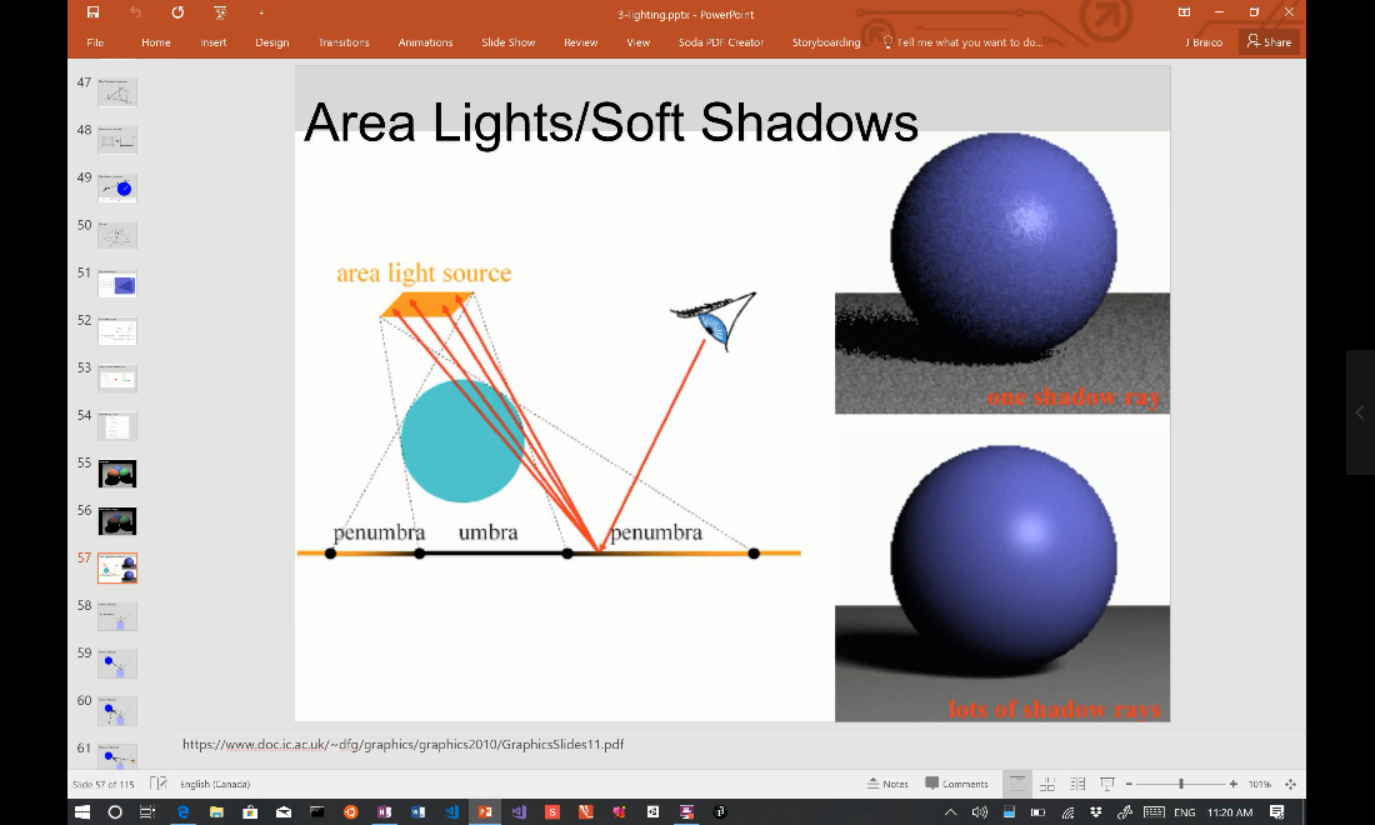
The program casts 4 rays instead of 1 ray into each pixel, and averages the 4 resulting sample colors. The averaged color is used as the color of that pixel.



(COMP4490-03\_20.mp4)

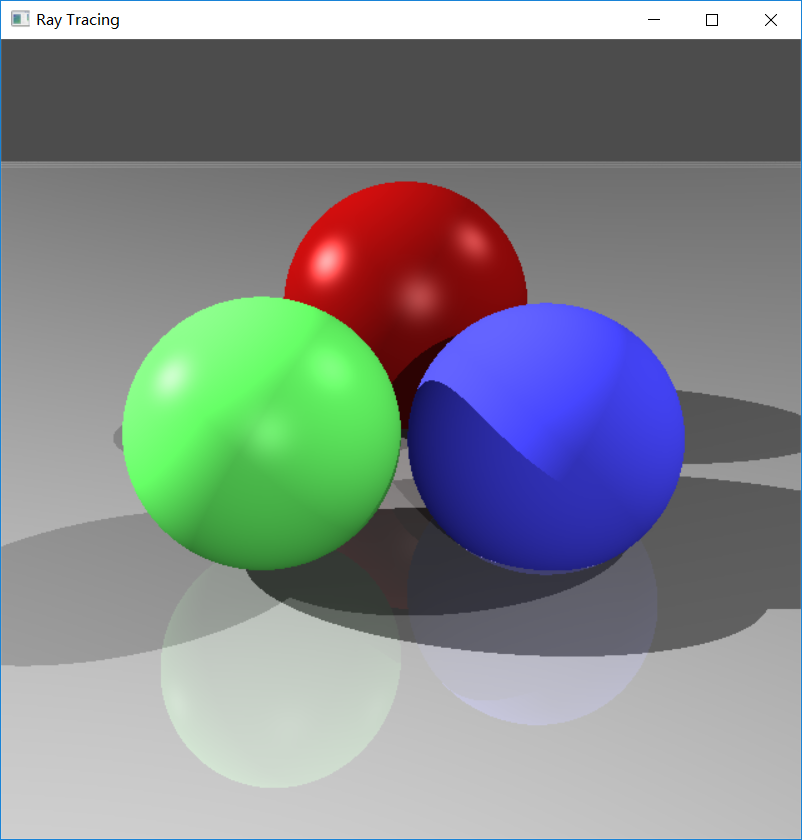
1. Area lighting.

The program considers an area light as a group of many point lights. For example, if the size of an area light is 10\*10, it is considered as 10 \* 10 = 100 point lights. Thus, to render a sample, the program just repeats the process for point light for 100 times, but each point light has an intensity of 1/100 of the overall intensity of the area light.



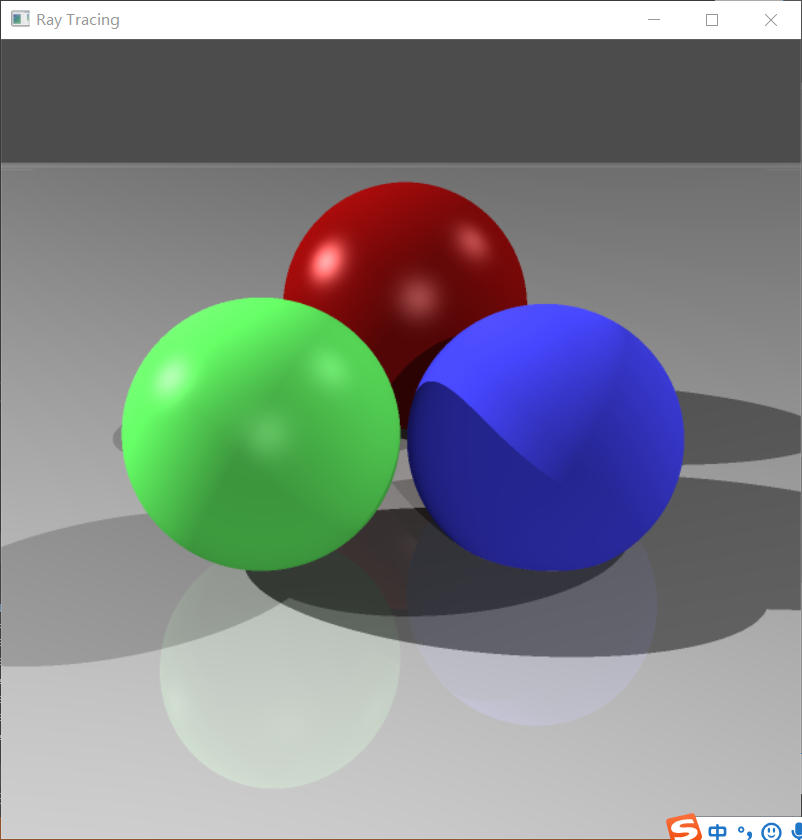
(COMP4490-03\_13.mp4)

**Outcome:**

****

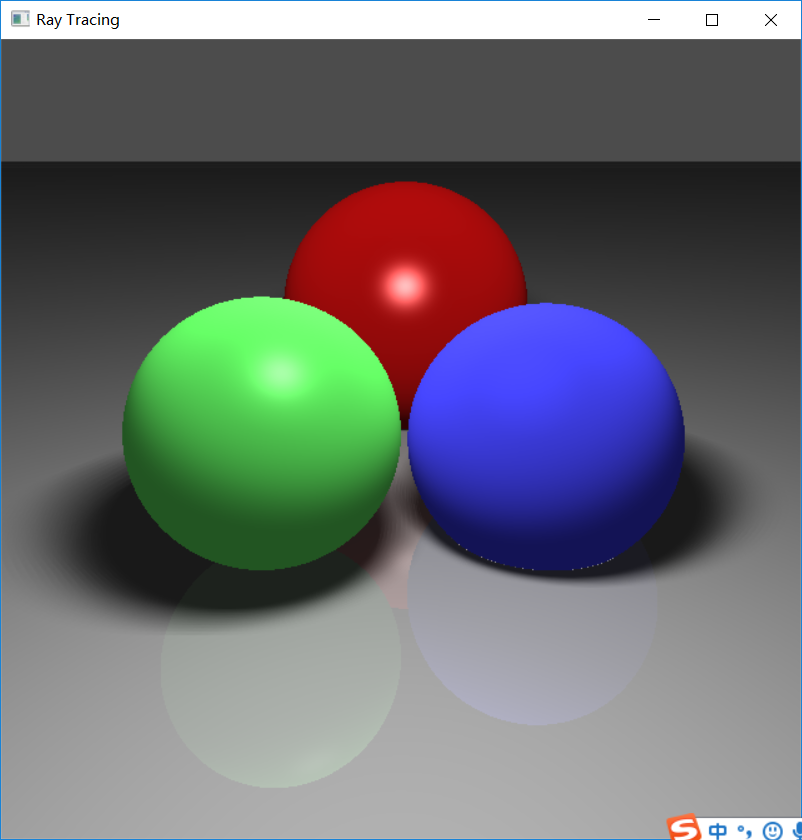
(Before applying antialiasing)

(d\_json\_origin.png)

****

(After applying antialiasing, the shadows are less jagged.)

(Feature5\_antialiasing\_d.json.png)

****

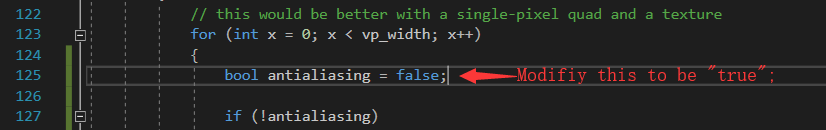
(After applying area lighting)

(Feature5\_areaLight\_m.json.png)

**How to test:**

1. Antialiasing:

For running speed’s sake, the antialiasing function is turned off by default. To test please modified “q1.cpp” as shown below:



Then please run d.json to test.

1. Area lighting:

Please run m.json to test.